Evaluating the Potential of Low Power Systems for Headphone-based Spatial Audio Applications

Jose A. Belloch[1]*, Alberto Gonzalez[2], Rafael Mayo[1], Antonio M. Vidal[3], and Enrique S. Quintana-Ortí[1]

[1] Depto. de Ingeniería y Ciencia de Computadores, Universitat Jaume I, 12071–Castellón, Spain
jbelloch@uji.es, mayo@uji.es, quintana@uji.es

[2] Depto. de Comunicaciones, Universitat Politècnica de València, 46022–Valencia, Spain
agonzal@dcom.upv.es

[3] Depto. de Sistemas Informaticos y Computación, Universitat Politècnica de València, 46022–Valencia, Spain
avidal@dsic.upv.es

Abstract

Embedded architectures have been traditionally designed tailored to perform a dedicated (specialized) function, and in general feature a limited amount of processing resources as well as exhibit very low power consumption. In this line, the recent introduction of systems-on-chip (SoC) composed of low power multicore processors, combined with a small graphics accelerator (or GPU), presents a notable increment of the computational capacity while partially retaining the appealing low power consumption of embedded systems. This paper analyzes the potential of these new hardware systems to accelerate applications that integrate spatial information into an immersive audiovisual virtual environment or into video games. Concretely, our work discusses the implementation and performance evaluation of a headphone-based spatial audio application on the Jetson TK1 development kit, a board equipped with a SoC comprising a quad-core ARM processor and an NVIDIA "Kepler" GPU. Our implementations exploit the hardware parallelism of both types of architectures by carefully adapting the underlying numerical computations. The experimental results show that the accelerated application is able to move up to 300 sound sources simultaneously in real time on this platform.

Keywords: Low Power Architectures, GPUs, Audio Processing, Spatial Audio, Fast Fourier Transform

1 Introduction

In the era of multimedia, smart phones and tablets, low power (embedded) processors play an important role for a myriad of applications. This is the case of improving the listener's

*Corresponding author. Thanks to projects TIN2011-23283, TEC2012-38142-C04-01, PROMETEO FASE II 2014/003, P1-1B2013-20 and EU FEDER

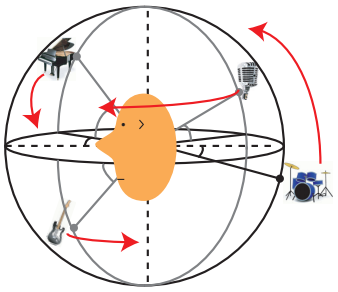

Figure 1: Headphone-based application that reproduces and moves multiple sound sources in real time.

experience via the development of sophisticated multichannel audio software [18] in general and headphone-based spatial audio applications in particular. Specifically, a spatial audio system based on headphones allows a listener to perceive the virtual position of a sound source [10]. This effect is obtained by processing sound samples through a collection of special filters that shape the sound with spatial information. In the frequency domain, these filters are known as *Head-Related Transfer Functions* (HRTFs), and the response of HRTFs describes how a sound wave is affected by properties of the individual's body (e.g. pinna, head, shoulders, neck and torso) before the sound reaches the listener's eardrum [13].

The computational problem that underlies a headphone-based application rapidly grows in cost with the number of sound sources that have to be handled (i.e., reproduced and moved) in real time while avoiding acoustic artifacts; see Figure 1. In past work [11, 12], we dealt with this computational complexity by leveraging a variety of high performance but power hungry graphics processing units (GPUs) connected to either a laptop or a desktop server (NVIDIA GTS360-M, GTX-580 and "Tesla" C2075 with 96, 448 and 512 CUDA cores, respectively).

Compared with our previous work, the main contribution of this paper is in the design of efficient implementations of a spatial audio application that exploit the specialized architecture of the Tegra K1 (TK1) systems-on-chip (SoC), embedded in the Jetson development kit (DevKit) [6]. This particular system comprises a quad-core ARM Cortex A15 processor (or CPU), an ARM Cortex A15 battery-saving shadow core, and an NVIDIA "Kepler" GPU with 192 CUDA cores. Therefore, the TK1 combines the luring low power consumption of embedded systems with the ample hardware parallelism of graphics accelerators, and adopted in mobile devices such as Google's Nexus 9 tablet [4]. Thus, one of the main features of the Jetson TK1 DevKit is it that allows a full-cycle integrated development of multimedia applications for mobile devices, from the initial design to the launching of the complete application.

The next section shows how the Jetson TK1 DevKit manage real-time audio processing. Section 3 describes the spatial sound application and analyzes the required operations that must be computed. Detailed GPU-based and ARM-based implementations of the audio application are presented in Section 4, where it is also shown a performance comparative. Finally, section 5 presents some concluding remarks.

2 Audio Processing on the Jetson TK1 DevKit

The Jetson TK1 DevKit (hereafter Jetson) is designed to accelerate the development of embedded software. For multimedia and audio applications, Jetson includes an ALC5639 Realtek HD

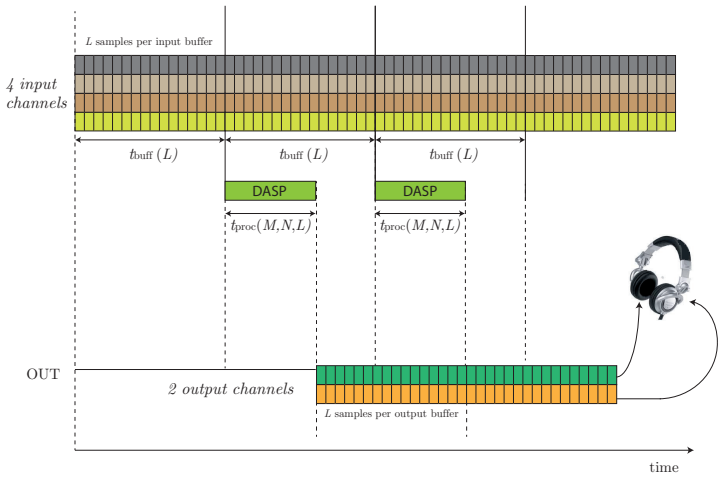

Figure 2: Operation of a real-time audio application.

Audio codec with *Mic in* and *Line out* jacks (routed to DAP2). This means that it is possible to reproduce two audio channels through the line out jacks and capture a signal through a microphone. The audio is managed in Jetson via the *Advanced Linux Sound Architecture* (ALSA) [1] software framework, which is a part of the Linux kernel, and provides an application programming interface (API) for sound card device drivers.

2.1 Real-Time performance

ALSA manages the communication between the input/output and the CPU using buffers of audio samples. Thus, for a sample frequency f_s=44,100 Hz (CD sound quality) and buffer sizes of L =1,024 samples, attaining real-time audio performance requires audio buffers of t_{buff} =23.22 ms (L/f_s) each. We note that this time is independent of the number of channels that are employed by the application.

For a real-time application, we have to take into account that ALSA requires/captures buffers of L samples per channel (input or output). Figure 2 shows how an audio application works with audio samples in real time. It is important to highlight the role of t_{proc}, which represents the time dedicated to process the M input buffers in order to obtain the N output buffers, where M and N are particularized in the figure to 4 and 2, respectively. As shown in the illustration, t_{proc} depends on M, N and L, while t_{buff} depends only on L. Figure 2 fits perfectly the target application, since there are only two output channels. The application works in real-time as long as $t_{\text{proc}} < t_{\text{buff}}$. Thus, the challenge consists in assessing how many sound sources can be spatially reproduced in real time by efficiently using all the computational resources available in Jetson.

3 Spatial Sound

A spatial effect can be achieved by convolving natural monophonic sounds that are recorded in an anechoic environment with a pair of filters that add spatial information from specific positions in the space to the audio wave. Figure 3 shows a sound source, in this case a piano,

that is located in the virtual position (θ, ϕ, r), where $\phi \in [-90°, +90°]$ represents the elevation coordinate, $\theta \in [0°, +360°]$ represents the azimuth coordinate, and $r \in [0, \infty]$ is the distance between the virtual position and the user.

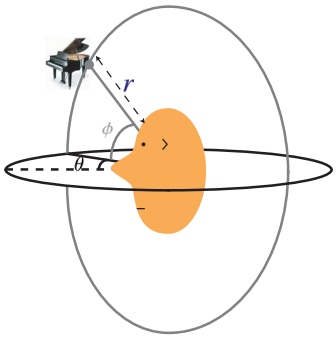

Figure 3: The HRIR filtering allows a person to perceive a piano sound as if it were located in a virtual position in the space given by the coordinates (θ, ϕ, r).

Two filters, one per ear, specify each virtual position. These are referred as *head-related impulse response* (HRIR) filters in the time domain and HRTF filters in the frequency domain. The HRIRs corresponding to position (θ, ϕ, r) in the time domain are denoted as $\mathbf{h}_{\mathrm{r}}(\theta, \phi, r)$ and $\mathbf{h}_{\mathrm{l}}(\theta, \phi, r)$ for the right and left ear, respectively.

Tailored HRTFs are usually obtained either through measurement and extrapolation [17] or via numerical simulation [14]. There are multiple public samples of HRTFs or HRIRs [5, 2]. In our case, we leverage the HRIR measures from [5]. These filters model a tall male with short hair, and can be found under measures `IRC_1007` in the website [5]. This HRIR database has azimuth and elevation resolutions, denoted by $\Delta\theta$ and $\Delta\phi$ respectively, representing the minimum separation in degrees between two positions of the database in azimuth and elevation. For our HRIR database, the resolution for both azimuth and elevation is 15°, and the distance of the sound source to the center of the head is fixed to r=1.95 m. Moreover all HRIR filters are windowed to a length of L=512 coefficients.

Let us employ $\mathbf{x}_{\mathrm{buff}}$ to denote an input-data buffer composed of L audio samples from a sound source x. Moreover, given a system composed of M sources, the input-data buffer $\mathbf{x}_{\mathrm{buff}_i}$ represents the buffer of L samples from source $i \in [0, M-1]$. The output-data buffers for the left and right ears, $\mathbf{y}_{\mathrm{buff}_l}$ and $\mathbf{y}_{\mathrm{buff}_r}$ respectively, are then given in the time domain by

$$\mathbf{y}_{\mathrm{buff}l} = \sum_{i=0}^{M-1} (\mathbf{h}_{\mathrm{l}}(\theta_i, \phi_i, r_i) * \mathbf{x}_{\mathrm{buff}i}) \quad \text{and} \quad \mathbf{y}_{\mathrm{buff}r} = \sum_{i=0}^{M-1} (\mathbf{h}_{\mathrm{r}}(\theta_i, \phi_i, r_i) * \mathbf{x}_{\mathrm{buff}i}), \tag{1}$$

where $*$ denotes the convolution operator.

The number of filters in the database limits the virtual positions to render. One complex problem is thus the synthesis of sound in virtual positions which do not belong to the collection of the filters. An additional difficulty occurs when the sound moves, which in practice requires the application of a new filter (i.e., a different HRIR). In particular, if the switch between HRIRs is not properly implemented, this may yield multiple audio clipping effects [15]. In order to address these two problems, it is possible to apply the audio techniques described next, to interpolate any position of the space and to switch properly between positions in the space.

3.1 The interpolation technique

This technique involves four convolutions as follows. Let us consider a generic position to render given by (θ_S, ϕ_S). From the specific elevation plane ϕ_S, the azimuth position to render θ_S is obtained by combining the rendered sound from the two nearby azimuth positions, θ_1 and θ_2, weighted by

$$w_A = \frac{\theta_S - \theta_1}{\Delta\theta} \quad \text{and} \quad w_B = \frac{\theta_2 - \theta_S}{\Delta\theta}; \tag{2}$$

see Figure 4. Then,

$$\mathbf{y}(\theta_S) = w_B \cdot \mathbf{y}(\theta_1) + w_A \cdot \mathbf{y}(\theta_2). \tag{3}$$

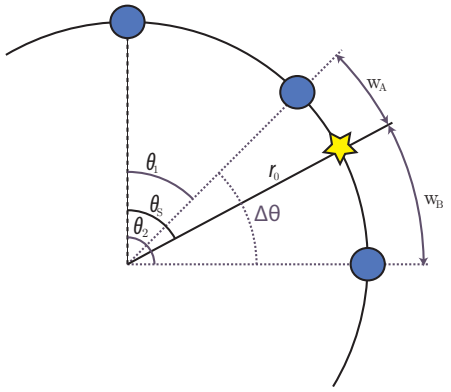

Figure 4: The star represents the position to be synthesized in the elevation plane ϕ_S. This position is synthesized by combining the two nearby azimuth positions using the weighted factors w_A and w_B.

Analogously, focusing on the specific azimuth plane θ_S, the elevation position to render ϕ_S is obtained by combining the rendered sound from the two nearby elevation positions, ϕ_1 and ϕ_2:

$$\mathbf{y}(\phi_S) = w_D \cdot \mathbf{y}(\phi_1) + w_C \cdot \mathbf{y}(\phi_2), \tag{4}$$

where the weighted factors are given by

$$w_C = \frac{\phi_S - \phi_1}{\Delta\phi} \quad \text{and} \quad w_D = \frac{\phi_2 - \phi_S}{\Delta\phi}. \tag{5}$$

Figure 4 also illustrates how to render a position with elevation ϕ_S, by simply substituting θ for ϕ and the resolution $\Delta\theta$ for $\Delta\phi$.

Combining (3) and (4), the rendered sound at virtual position (θ_S, ϕ_S) is then given by

$$\mathbf{y}(\theta_S, \phi_S) = w_D \cdot (w_B \cdot \mathbf{y}(\theta_1, \phi_1) + w_A \cdot \mathbf{y}(\theta_2, \phi_1)) + w_C \cdot (w_B \cdot \mathbf{y}(\theta_1, \phi_2) + w_A \cdot \mathbf{y}(\theta_2, \phi_2)). \tag{6}$$

3.2 The switching technique

This is used mainly for the virtualization of source movement, which is carried out in this application by smoothly varying the virtual positions of the source over time. For example, assume the sound source x_i moves from position A: (θ_{SA}, ϕ_{SA}) to position B: (θ_{SB}, ϕ_{SB}). In practice, this requires to switch the reproduction between the two positions. However, this

switching could produce multiple audible clipping effects if it is not properly executed. The switching technique that we employ to reduce possible artifacts is described in [15]. That work suggests the application of a fading, which is a gradual increase in the sound filtered by position B while the sound filtered by position A decreases in the same proportion. To this end, the current input-data buffer $\mathbf{x}_{\text{buff}_i}$ must be computed for both positions, and then the result of the two computations must be multiplied element-wise by two fading vectors, say $\mathbf{f}$ and $\mathbf{g}$. Finally, the output-data buffer $\mathbf{y}_{\text{buff}_i}$ is obtained by summing the results from the two previous multiplications element-wise:

$$\mathbf{y}_{\text{buff}i}(\theta_S, \phi_S) = ((\mathbf{y}_{\text{buff}_i}(\theta_{SB}, \phi_{SB}) \otimes \mathbf{f}) \oplus ((\mathbf{y}_{\text{buff}_i}(\theta_{SA}, \phi_{SA}) \otimes \mathbf{g}). \quad (7)$$

where $\otimes$ and $\oplus$ respectively represent the element-wise multiplication and addition operators.

4 Implementations and Experimental Evaluation

Once the positions of each sound source has been selected and the input-data buffers are received, the processing to carry out in the application is summarized as:

Algorithm 1 Headphone-based spatial audio application

Input: M, $\mathbf{x}_{\text{buff}}$, $\mathbf{h}(\theta, \phi)$
Output: $\mathbf{y}_{\text{buff}r}$ $\mathbf{y}_{\text{buff}l}$

1: **for** $e = \{l, r\}$ **do** ▷*For the Left and Right ear*
2: **for** $m = 0, \ldots, M-1$ **do** ▷*For each sound source*
3: **for** $p = \{A, B\}$ **do** ▷*Two positions for each sound source*
4: **for** $k = \{(\phi_1, \theta_1), (\phi_1, \theta_2), (\phi_2, \theta_1), (\phi_2, \theta_2)\}$ **do** ▷*The four interpolated positions*
5: $\mathbf{y}_{\text{buff}_{ep}}(\theta_p, \phi_p) = \mathbf{y}_{\text{buff}_{ep}}(\theta_p, \phi_p) + \mathbf{h}_e(\theta_{kp}, \phi_{kp}) * \mathbf{x}_{\text{buff}m}$.
6: **end for**
7: **end for**
8: $\mathbf{y}_{\text{buff}_e} = ((\mathbf{y}_{\text{buff}_{eB}}(\theta_{eB}, \phi_{eB}) \otimes \mathbf{f}) \oplus ((\mathbf{y}_{\text{buff}_{eA}}(\theta_{eA}, \phi_{eA}) \otimes \mathbf{g})$.
9: **end for**
10: **end for**

The convolution operation is carried out in real time by using the overlap-save technique with a 50% overlap in the frequency domain [16]. That means that convolution is converted in an element-wise multiplication of two vectors of size $2L$ composed of complex values. Considering the operations described in the previous section, we can conclude that for synthesizing $\mathbf{y}_{\text{buff}l}$ and $\mathbf{y}_{\text{buff}r}$, it is required to compute 16 convolutions per sound source. Table 1 enumerates the operations that are executed by a headphone-based spatial audio application that reproduces M sound sources

Note that only the input-data buffers $\mathbf{x}_{\text{buff}i}$ must be transformed to the frequency domain since the filters HRTF are already in the frequency domain. The following subsections present the implementation of a complete headphone audio application on the two computational kernels of the Jetson: a GPU Nvidia Kepler with 192 cuda cores and a CPU ARM Cortex-A15 with four cores.

4.1 GPU-based implementation

The GPU that is embedded in the TK1 processor adheres to the Kepler GPU architecture [7] and is virtually identical (obviously, scaled down) to those present in high-end systems. The

Step Number	*Number of computations*	*Operation description*
1	M	FFT of size $2L$
2	16M	1 Element-wise Multiplications between two complex vectors of size $2L$
3	4	4M Element-wise Sum between two complex vectors of size $2L$, see (6)
4	4	IFFT of size $2L$
5	2	2 Element-wise Multiplications between two real vectors of size L
6	2	2 Element-wise Sum between two real vectors of size L, see (7)

Table 1: Operations that are required by the spatial audio application.

Kepler GPU in the TK1 consists of a single streaming multiprocessor (SMX) composed of 192 CUDA cores. In CUDA, the GPU code is written as a kernel function, to be concurrently executed by multiple threads grouped into thread blocks. An important aspect to consider is that all threads inside the same block can access data in a coalesced way.

Regarding the steps 1) and 4) from Table 1, NVIDIA provides a specialized FFT library [8] that we leveraged in our GPU routine to compute multiple 1-D FFTs. In addition, Steps 2)–3) and 5)–6) are implemented respectively as the *ad hoc* Kernels 1 and 2 described next.

4.1.1 Kernel 1

This kernel computes $\mathbf{Y}(\theta_{SA}, \phi_{SA})$ and $\mathbf{Y}(\theta_{SB}, \phi_{SB})$ in the frequency domain, derived from the time domain formulation in (6), as

$$\mathbf{Y}(\theta_S, \phi_S) = w_D \cdot w_B \cdot \mathbf{Y}(\theta_1, \phi_1) + w_D \cdot w_A \cdot \mathbf{Y}(\theta_2, \phi_1) + w_C \cdot w_B \cdot \mathbf{Y}(\theta_1, \phi_2) + w_A \cdot w_C \cdot \mathbf{Y}(\theta_2, \phi_2). \quad (8)$$

Although there exists ample concurrency in this computation, we note that $\mathbf{Y}(\theta, \phi)$ are vectors of size $2L$ with complex entries. The CUDA capability of the TK1 (3.2) enforces a maximum number of 16 resident blocks per multiprocessor. Therefore, with only one SMX in the TK1, we execute a kernel composed of 16 thread blocks, where each block comprises $4L/16$ threads, an each thread is devoted to compute one complex sample of $\mathbf{Y}(\theta_{SA}, \phi_{SA})$ and $\mathbf{Y}(\theta_{SB}, \phi_{SB})$; see Figure 5. In more detail, each thread carries out $8M$ times the operations:

$$\begin{aligned} res.r &= res.r + (h.r \cdot sig.r - h.i \cdot sig.i) \cdot w_1 \cdot w_2, \\ res.i &= res.i + (h.r \cdot sig.i - h.i \cdot sig.r) \cdot w_1 \cdot w_2, \end{aligned} \quad (9)$$

where res is a complex sample of $\mathbf{Y}(\theta_S, \phi_S)$; h represents a coefficient of a HRTF filter; sig denotes a complex sample of the input signal; both w_1 and $w_2 \in \{w_A, w_B, w_C, w_D\}$; and the suffixes $.r$ and $.i$ denote the real and imaginary parts of a complex value. This implementation of the kernel requires the use of 36 registers per thread.

4.1.2 Kernel 2

This kernel computes $\mathbf{y}(\theta_{SA}, \phi_{SA})$ and $\mathbf{y}(\theta_{SB}, \phi_{SB})$ from (7). Following the same guidelines exposed for Kernel 1, we execute a kernel with 16 thread blocks, each block composed of $2L/16$ threads, and each thread devoted to compute one complex sample of the output; see Figure 6. In this case, the overlapping samples are deleted according to the convolution theorem [16]. This kernel uses 8 registers per thread.

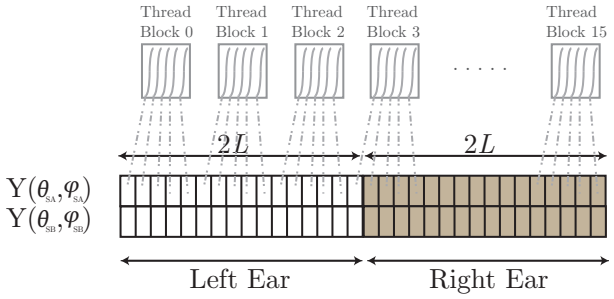

Figure 5: Each thread computes one complex sample of $\mathbf{Y}(\theta_S A, \phi_S A)$ and of $\mathbf{Y}(\theta_S B, \phi_S B)$ in Kernel 1.

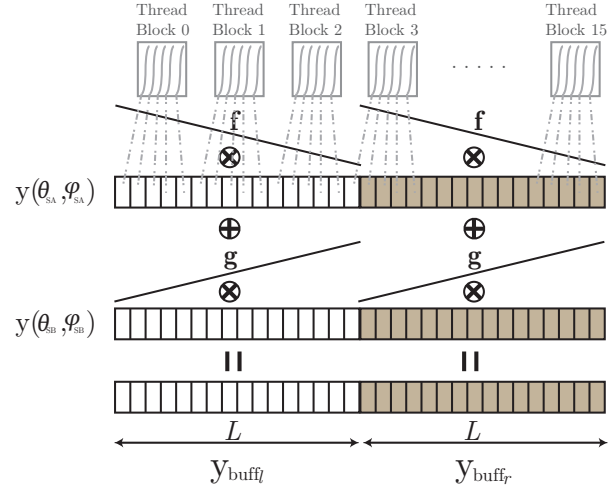

Figure 6: Each thread computes one complex sample of $\mathbf{y}_{\text{buff}l}$ and $\mathbf{y}_{\text{buff}r}$ in Kernel 2.

4.2 ARM-based implementation

The TK1 processor also contains a low power ARM Cortex-A15 with four cores. In order to exploit this type of hardware concurrency, we used *openMP* [9] to parallelize the appropriate computations. Steps 1) and 3) require the use of a FFT library and, to this end, we selected the implementation in the *Fast Fourier Transform West* [3] which can be used on ARM processors in combination with *openMP*.

Step 5) involves three nested loops, which iterate over the samples of the buffer ($4L$, because of the left and the right ear), the sound sources ($2M$, because of the switching technique), and the sound sources (4 because of the interpolation technique). Furthermore, the operation that must be computed at the loop body is given in (9). We implemented our algorithm using *openMP*, partitioning the iteration space of the outermost loop among the four cores via the OpenMP directives `omp_set_num_threads(4)` and `#pragma omp parallel for schedule(static,1)`. The same methodology was followed for step 6), but in this case there is only the loop that references the samples of the output buffers ($2L$). Finally, the functions that implemented steps 5) and 6) were compiled using auto-vectorization flags: `-mfpu=neon -ftree-vectorize`.

4.3 Performance evaluation

The objective of the following experiment is to assess the highest number of sound sources M that can be reproduced in real time using the new GPU and ARM implementations for the TK1 processor in Jetson. The computational experiment thus consisted of launching the application and gradually increasing the number of sources. For each execution, the time t_{proc} was measured and compared with the threshold t_{buff}. As explained in Section 2.1, the application attains real time if $t_{\text{proc}} < t_{\text{buff}}$. Otherwise, the application can still be considered as an off-line technology. Figure 7 exposes the evolution of t_{proc} as a function of the number of sources for three different implementations: GPU code; sequential ARM code using a single Cortex A15 core/thread; and multithreaded ARM using the four Cortex A15 cores. The time t_{buff} that symbolizes the border between real-time and off-line applications is also shown as a flat line.

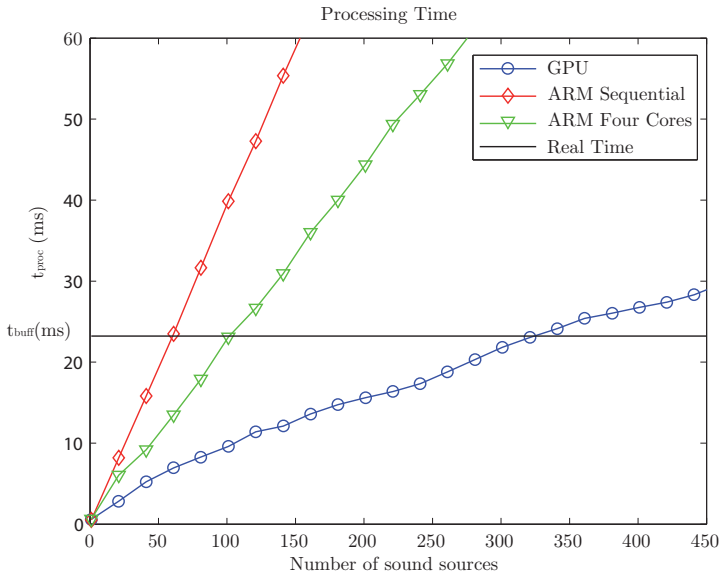

Figure 7: Number of sound sources that can be handled in real time using the computational resources of the Jetson TK1 DevKit.

The GPU-based implementation can interact with 325 sound sources in real time, which is a huge number of sound sources. The ARM CPU also delivers a considerable performance, and it can reproduce up to 55 using only one core and up to 100 sound sources with the four cores. The lack of scalability for the latter is due to the internal implementation of the FFTW library, which offers a limited speed-up.

5 Conclusion

Tablets and smart phones are nowadays equipped with powerful yet low power processors. This is the case of the NVIDIA Tegra K1 mobile processor that is present, among others, in the google nexus 9 tablet. This new developments paves a captivating road for a vast amount of applications that require high computational resources and which can then now be executed in this type of architectures for mobile appliances. In this work, we have assessed the performance

of the Jetson TK1 board for an application that synthesizes spatial sound in real time. To this end, we have implemented and optimized the application for two target architectures: the 192-core GPU and the quad-core ARM Cortex A15. Our experiments show that the GPU in this system is able to reproduce up to three times more sound sources than the ARM CPU, but also that both can handle a remarkably high number of sound sources in real time.

References

[1] Advanced linux sound architecture (alsa):. `http://www.alsa-project.org/`. (accessed 2015 January 08).

[2] The CIPIC HRTF Database. *online at: http://interface.cipic.ucdavis.edu/sound/hrtf.html*.

[3] Fast Fourier Transform West. `http://www.fftw.org/`. (accessed 2015 January 11).

[4] Google's nexus 9. `http://blogs.nvidia.com/blog/2014/10/17/nvidia-tegra-k1-google-nexus-9/`. (accessed 2015 January 11).

[5] Listen HRTF database. *online at: http://recherche.ircam.fr/equipes/salles/listen/index.html*.

[6] Mobile GPU: Jetson. `http://developer.download.nvidia.com/embedded/jetson/TK1/docs/Jetson_platform_brief_May2014.pdf`. (accessed 2014 November 22).

[7] Nvidia CUDA Developer Zone. `https://developer.nvidia.com/cuda-zone`. (accessed 2014 April 10).

[8] Nvidia FFT library. `https://developer.nvidia.com/cuFFT`. (accessed 2015 January 09).

[9] openMP API Specifications. `http://www.openmp.org`. (accessed 2013 June 05).

[10] V.R. Algazi and R.O. Duda. Headphone-based spatial sound. *IEEE Signal Processing Magazine*, 28(1):33–42, 2011.

[11] Jose A. Belloch, M. Ferrer, Alberto González, F.J. Martínez-Zaldívar, and A. M. Vidal. Headphone-based spatial sound with a GPU accelerator. *Procedia Computer Science*, 9(0):116–125, 2012.

[12] Jose A. Belloch, M. Ferrer, Alberto González, F.J. Martínez-Zaldívar, and A. M. Vidal. Headphone-based virtual spatialization of sound with a GPU accelerator. *Journal of the Audio Engineering Society*, 61(7/8):546–561, 2013.

[13] Jens Blauert. *Spatial Hearing - Revised Edition: The Psychophysics of Human Sound Localization*. The MIT Press, 1996.

[14] Yuvi Kahana and Philip A. Nelson. Numerical modelling of the spatial acoustic response of the human pinna. *Journal of Sound and Vibration*, 292(1-2):148 – 178, 2006.

[15] Akihiro Kudo, Haruhide Hokari, and Shoji Shimada. A study on switching of the transfer functions focusing on sound quality. *Acoustical Science and Technology*, 26(3):267–278, 2005.

[16] A. V. Oppenheim, A. S. Willsky, and S. Hamid. Signals and systems. Processing series. Prentice Hall, 2nd edition, 1997.

[17] S. Spors and J. Ahrens. Efficient range extrapolation of head-related impulse responses by Wave Field Synthesis techniques. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, Prague, Czech Republic, May 2011.

[18] E. Torick. Highlights in the history of multichannel sound. *J. Audio. Eng. Soc.*, 46(5):27–31, 1998.